arXiv:2610.34436v1 [cs.DC] 25 Oct 2026

PREFIX KEYS AND VALUES MATTER LATE: RIPE CACHING FOR SELF-SPECULATIVE DECODING IN LOOPED LANGUAGE MODELS

Dezhen Lu [1] **Runguo Li** [2] **Zixiang Ni** [3]

ABSTRACT

A looped language model runs one block of layers R times per token, so its key–value (KV) cache holds R planes per recurrent layer, and self-speculative decoding (draft with a few loops, verify with all R) reads them all. We find that prefix keys and values matter late: at equal perturbation size, loops 25–32 are $e^{15.0}$ (Huginn-0125) and $e^{12.8}$ (Raven-Llama) times more sensitive than loops 2–9, and the planes approach the last one geometrically, at 0.88 and 0.81 per loop. Whether the deepest plane alone suffices then depends on how ripe the late planes are: on Raven, storing only it is indistinguishable from a bf16-to-fp32 change; on Huginn, 1% of positions carry 74% of its error. We turn this into RIPE, which stores the deepest plane everywhere except a window of recent positions kept at full depth. On Huginn, a 64-position window removes 89% of the tail positions at 11.9% of the full cache's memory (1024-token context), and any window at least as long as the draft block makes batched verification block-independent by construction. Nine pre-registered GPU batches back these results. Simple models of acceptance, decoding cost and memory, two of them checked on held-out measurements, account for the measured self-speculative speed-ups (2.09× with the full cache, 2.58× with the deepest plane) and decompose the 2.24× fixed-memory throughput gain of one shared plane.

1 INTRODUCTION

Looped, or recurrent-depth, language models reuse one block of transformer layers many times per token (Geiping et al., 2025; Zhu et al., 2025; McLeish et al., 2025). Because a shallow run of the same weights can serve as a draft, they support self-speculative decoding without a separate draft model: draft with $d \ll R$ loops, verify with all R (Geiping et al., 2025; Cho et al., 2026; Ha & Kim, 2026; Li et al., 2026; Liu et al., 2026). The weights are small; the KV cache is not. Every loop of every recurrent layer writes its own keys and values, so Huginn-0125 stores 2,788 KB per token, against 169 KB if only one plane were kept. At an 808-token sequence, a 95 GB budget then holds 36 sequences with the full cache and 302 with one plane (Section 3.5).

Which planes have to be stored? Huginn's own code offers caches that keep fewer planes (Geiping et al., 2025), Ouro reports that the last loop's KV suffices for its model (Zhu et al., 2025), and WaveFront shows that cross-loop sharing raises self-speculative speed (Ha & Kim, 2026). We measured the question directly in nine pre-registered GPU batches: decision rules and predictions were frozen before each run, and many predictions failed (Appendix B lists all of them). This paper organizes those measurements with small analytical models (Table 1 lists what each was fitted on and which checks use held-out data) and turns them into a cache design.

RIPE in one line: *ripe plane for the past, full depth for the present* (Figure 1).

Contributions.

- *Where KV precision matters.* A sensitivity model (KL grows as a power, exponent 1.82–2.70, of a prefix-KV perturbation) and a ripening model (planes converge geometrically with depth) show that only the late loops need accurate prefix keys and values, and predict when the deepest plane alone costs no more than bf16 rounding (Sections 3.1, 3.2). Truncating finished tokens to shallow planes, our starting plan, is backwards.
- *RIPE.* A cache that keeps the deepest plane everywhere except a recent full-depth window, with its memory cost, a decoding step (Algorithm 1) and a proof that batched verification is block-independent when the window covers the block (Section 4).
- *Pre-registered evidence on two model families.* Nine GPU batches with frozen decision rules, including every failed prediction: the deepest plane's error tail on Huginn and its removal by a 64-position window, its absence on Raven, and real self-speculative speed-ups

[1]University of Tübingen, Tübingen, Germany [2]University of Illinois Urbana-Champaign, Urbana, IL, USA [3]Xi'an Jiaotong University, Xi'an, China. Correspondence to: Dezhen Lu <dezhen.lu@student.uni-tuebingen.de>, Runguo Li <runguol2@illinois.edu>, Zixiang Ni <nzx@stu.xjtu.edu.cn>.

Preprint.

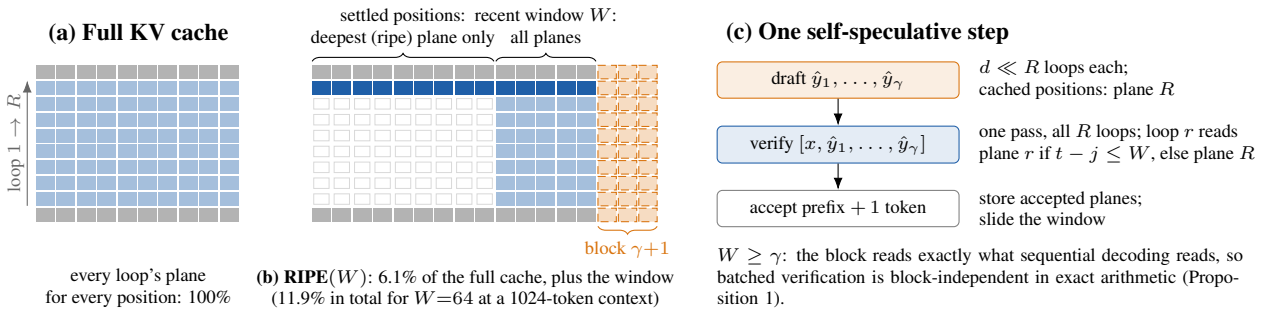


Figure 1. RIPE at a glance (schematic; grid sizes not to scale). Rows are the prelude (bottom), loops $1, \ldots, R$ and the coda (top) of each prefix position (columns). (a) A full cache keeps every loop's keys and values. (b) RIPE keeps only the deepest, ripe plane for settled positions and all planes for the W most recent ones; the block being verified lies inside the window. Memory from Eqs. (3) and (10). (c) A self-speculative step. We measured the deepest-plane cache ($W = 0$) inside the speculative decoder and windows $W > 0$ by teacher-forced replay (Section 7).

(Section 5).

- *Performance models and pitfalls.* Acceptance, cost, memory and fixed-memory throughput models, two checked on held-out data (Sections 3.3–3.5); per-head position filtering, sinks and a second plane do not help, and cuDNN attention makes bf16 decoding non-reproducible on GH200 (Sections 5.4, 5.5).

2 Background

Looped models. A prelude P of n_{pre} layers embeds token x_t; a core block B of n_{core} layers runs R times on a state initialized at random; a coda C of n_{coda} layers produces logits:

$$s_t^{(r)} = B\big(s_t^{(r-1)},\, P(x_t);\, \mathcal{C}_{<t}^{(r)}\big), \qquad r = 1, \ldots, R, \quad (1)$$

$$z_t = C\big(s_t^{(R)}\big). \quad (2)$$

At loop r, each core layer emits keys and values for token t; we call loop r's keys and values over all core layers *plane* r, written $(K^{(r)}, V^{(r)})$. In the full cache, the query of token t at loop r attends to plane r of every prefix position. Huginn-0125 (Geiping et al., 2025) has $n_{\text{pre}} = n_{\text{coda}} = 2$, $n_{\text{core}} = 4$, $R = 32$, and 55 heads of width 96. Raven-Llama (McLeish et al., 2025) has $n_{\text{pre}} = n_{\text{coda}} = 4$, $n_{\text{core}} = 6$, $R = 32$, and grouped-query attention with 8 KV heads of width 64.

Cache policies. A policy $\pi(r, t, j) \in \{1, \ldots, R\}$ names the plane of prefix position j that token t reads at loop r (Figure 1). The full cache has $\pi = r$; the *deepest-plane* cache has $\pi = R$. Huginn's `compress-sS` keeps S core slots and overwrites them modulo S; with $S = n_{\text{core}}$ the last writer of every slot is loop R, so it is exactly the deepest-plane cache. Huginn's slot map hard-codes $n_{\text{pre}} = 2$; for Raven we use the general map slot $= (\text{step} - n_{\text{pre}}) \bmod S + n_{\text{pre}}$, checked on CPU against direct substitution. With $b = 2\, H_{\text{kv}}\, d_h$ bytes per layer per token in bf16, an L-token sequence needs

$$M_{\text{full}}(L) = bL\,(n_{\text{pre}} + n_{\text{coda}} + R\, n_{\text{core}}), \quad (3)$$

$$M_{\text{deep}}(L) = bL\,(n_{\text{pre}} + n_{\text{coda}} + n_{\text{core}}),$$

that is, 2,788 vs. 169 KB per token on Huginn (6.1%) and 410 vs. 28.7 KB on Raven (7.0%).

Self-speculative decoding. A draft runs γ tokens through d loops each; one verification pass runs the block of $\gamma + 1$ tokens through all R loops at once, accepts the longest prefix on which draft and verifier agree (greedy), and adds one token from the verifier. We write τ for tokens per verification.

Measurement modes. *Teacher-forced prefill* replaces attention with an explicit fp32 softmax, records each call's keys and values, and substitutes any loop's prefix K/V by another plane; the last 64 positions act as decoded tokens. *Generation* decodes greedily with the full cache, then replays each trajectory through the arm and through an fp32 full cache, which sets the precision floor. We report $\mathrm{KL}(\text{reference} \,\|\, \text{arm})$ per position.

3 Models

All models in this section are post hoc: we built them after the pre-registered batches had run. We label each check by whether the data it uses entered the fit.

3.1 Sensitivity: only the deep loops' prefix KV matters

Let δ_r be the relative perturbation of the prefix keys and values that loop r reads. To first order the logits move by $\Delta z \approx \sum_r J_r \delta_r$. With $p = \mathrm{softmax}(z)$ and $p' = \mathrm{softmax}(z + \Delta z)$,

$$\mathrm{KL}(p \,\|\, p') = \tfrac{1}{2}\, \Delta z^\top F(p)\, \Delta z + O(\|\Delta z\|^3), \quad (4)$$

Table 1. Post hoc models of this paper and how each was checked. "Fitted on" names the data that set its parameters; "Check" says whether the comparison data were held out from the fit. Batch numbers refer to Appendix A.

Model	Form	Fitted on	Checked against	Check	Result
KV memory	Eq. (3), (10)	architecture only	26 WaveFront cells; window extras (b. 8)	held out	≤ 0.005 GB (print precision); exact
Peak memory	cache $+ o_0 + o_1 B$	batch-6 cells	batch-8 cells, other node, $B \leq 256$	held out	$\leq 0.15\%$
Sensitivity	$\log \mathrm{KL} = a + b[\text{deep}] + g \log \lVert\delta\rVert$	region arms (pre-registered)	second-order theory, $g = 2$	theory	$g = 2.70$ / 1.82 (fp32)
Ripening	$\Delta_r = c\rho^r$; bf16 adds Δ_∞	fp32 curve, loops 8–31	fp32 curve; bf16 curve with one more parameter	in sample	max $\lvert\log$ err$\rvert$ fp32 0.28 / 0.51, bf16 0.27 / 0.19
Deepest-plane error	Eq. (6)	sensitivity + ripening	deepest-plane cache arm (fp32), not in the fit	partly held out	3.6×10^{-6} vs. 3.2×10^{-6} (factor 2)
Acceptance	$\tau = (1 - \alpha^{\gamma+1})/(1 - \alpha)$	batch-3 simulation	batch-8 real decoder	held out	within 6%
Speed-up	Eq. (8)	one factor η, 24 timings	the same timings, both caches	in sample	2.08/2.61 vs. 2.09/2.58

where $F(p) = \mathrm{diag}(p) - pp^\top$. Dropping cross terms, $\mathrm{KL} \approx \sum_r s_r \lVert\delta_r\rVert^2$ for small perturbations: quadratic, weighted by a per-loop sensitivity s_r. We fit the exponent rather than fix it at 2.

The pre-registered test of batches 2 and 9 substitutes the prefix K/V of either loops 2–9 (shallow) or loops 25–32 (deep) by plane $p \in \{1, 4, 8, 16, 24, 32\}$, measures the size $\lVert\delta\rVert$ as the root-sum-square of the relative distances over loops, layers, keys and values, and regresses $\log \mathrm{KL}$ at the first decoded position on region and $\log \lVert\delta\rVert$ (Figure 2a,b). In fp32 the size exponent is 1.82 [1.68, 1.97] on Raven, consistent with the second-order term, and $g = 2.70$ [2.52, 2.87] on Huginn, which excludes 2: Huginn's shallow loops follow the quadratic (slope 2.06) but its deep loops are steeper (2.71). The largest substitutions (plane 1) are outside the small-perturbation regime of Eq. (4); Raven's shallow region sits at the fp32 resolution. At equal size the deep region is $e^{15.0}$ [$e^{14.2}$, $e^{15.8}$] (Huginn, 40 texts) and $e^{12.8}$ [$e^{12.0}$, $e^{13.7}$] (Raven, 40 texts) times more sensitive. In bf16 the coefficients are 6.7 and 3.8; there 23% and 21% of the rows sit at the bf16 KL floor, which caps the measurable gap.

The consequence for caching: the loops that need accurate prefix K/V are the deep ones. At equal storage, keeping the deepest plane agrees with the full cache at the first decoded position in 0.990 of texts, whereas a window of the first four loops, the starting plan's choice, reaches 0.845 (Table 3).

3.2 Ripening: planes converge geometrically

Let $\Delta_r = \lVert K^{(r)} - K^{(R)}\rVert / \lVert K^{(R)}\rVert$ over prefix positions, averaged over layers and texts, and likewise for values. Figure 2c shows that in fp32, $\Delta_r \approx c\rho^r$ from loop 8 on (on Raven the rate drifts upward with r; largest log residual 0.51). Huginn ripens at $\rho = 0.88$ per loop (half-life 5.2 loops) and Raven at $\rho = 0.81$ (3.3 loops); values behave the same way ($\rho = 0.87$ and 0.81). In bf16 the curve flattens at a rounding floor,

$$\Delta_r^{\mathrm{bf16}} \approx \sqrt{(c\rho^r)^2 + \Delta_\infty^2}, \tag{5}$$

and this one extra parameter reproduces the bf16 curve ($\Delta_\infty = 0.0072$ for Huginn's keys, 0.0048 for Raven's). Over the deep loops 25–31, Huginn's keys are still 0.0116 away from plane R in bf16, above its floor, while Raven's are at its floor (0.0047); in fp32 Raven's are 61 times closer than Huginn's.

Deepest-plane error. Storing only plane R sets $\delta_r = \Delta_r$ for every $r < R$. With the sensitivity profile of Section 3.1, only the deep loops count, and the fitted region model gives

$$\mathrm{KL}_{\mathrm{deep}} \approx e^{a+b}\, S^g, \qquad S^2 = n_{\mathrm{core}} \sum_{r=25}^{31} \left(\Delta_r^{K\,2} + \Delta_r^{V\,2}\right). \tag{6}$$

For Huginn in fp32 this predicts 3.6×10^{-6} at the first decoded position from the ripening curve, or 1.7×10^{-6} from the measured perturbation size; the deepest-plane cache, which the fit did not use, measures 3.2×10^{-6} (median). The agreement is to within a factor of two, not better. Substituting plane R in the shallow loops alone costs only 2.4×10^{-9}. For Raven the model predicts 1.8×10^{-8} (4.7×10^{-9} from the measured size), 202–366 times less than for Huginn, and the measurement is at the fp32 resolution (median below 10^{-9}, mean 2.7×10^{-8}).

Remark 1 (ripeness test, a hypothesis). If the deep loops'

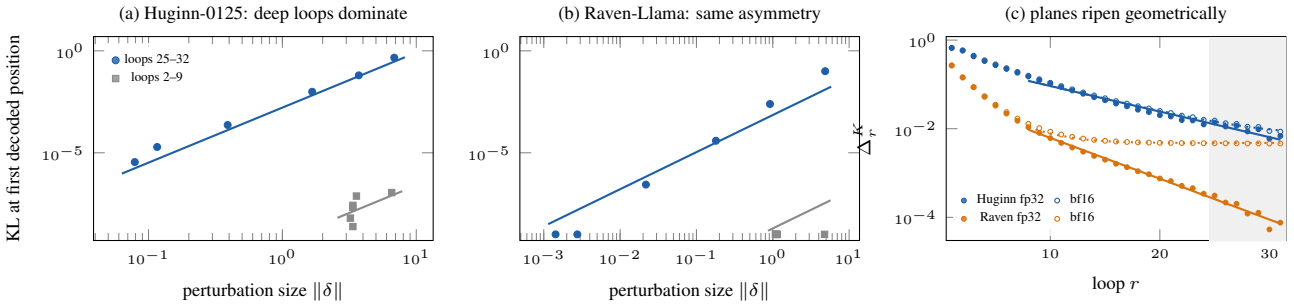


Figure 2. Why only late planes matter, and when the deepest plane is enough. (a,b) First-decoded-position KL after substituting the prefix keys and values of loops 2–9 or 25–32 by planes 1–32 (fp32, AlpacaEval; points: arm medians over texts; lines: the pre-registered regression). At equal perturbation size the deep loops cost $e^{12.8}$–$e^{15.0}$ times more. (c) Distance of loop r's prefix keys to loop R's: filled fp32, open bf16; solid $c\rho^r$ fitted to fp32, dotted with one bf16 floor (Eq. (5)). Raven's deep loops (shaded) sit at the bf16 floor; Huginn's do not.

Δ_r is at the bf16 floor, deepest-plane storage perturbs those loops about as much as rounding already does, and its cost should be indistinguishable from a precision change; otherwise it should exceed it. Both models fit this reading (Section 5.1), but two models cannot establish it. The test needs one prefill per prompt and no generation.

3.3 Acceptance

If each draft token is accepted with probability α, independently of its position in the block, the tokens per verification follow (Leviathan et al., 2022; Chen et al., 2023)

$$\tau(\alpha,\gamma)=\frac{1-\alpha^{\gamma+1}}{1-\alpha}. \tag{7}$$

Batch 3 simulated greedy self-speculation on 60 GSM8K problems with $\gamma=4$ and recorded acceptance at each draft position given that all earlier ones were accepted. It is flat: across positions it varies by at most 0.06 (full cache) and 0.04 (deepest plane) for $d\in\{2,4,8\}$ (Figure 5a), and Eq. (7) with the mean α_d matches the simulated τ within 2.6%. Out of sample, the batch-3 values predict $\tau=3.40$ (full) and 4.33 (deepest plane) at $d=\gamma=4$; batch 8's real batched-verify decoder measured 3.34 and 4.47 on 24 other GSM8K problems and 3.61 and 4.53 on 12 more. Drafts that read the deepest plane accept $\alpha_4=0.93$ instead of 0.81, and their rejection rate falls faster with draft depth: $1-\alpha_d\propto\rho_a^d$ with $\rho_a=0.66$ against 0.76.

3.4 Cost

We assume that at batch size 1 our eager decoder is bound by kernel launches and weight reads, so that a forward costs in proportion to the layer passes it runs, $c(d)=n_{\text{pre}}+d\,n_{\text{core}}+n_{\text{coda}}$, almost regardless of how many tokens it carries. A sequential token costs $c(R)=132$ passes; a block costs $\gamma\,c(d)+c(R)$, scaled by one overhead factor η for everything else. The speed-up over sequential decoding is then

$$S(d,\gamma)=\frac{\tau(\alpha_d,\gamma)\;c(R)}{\eta\left(\gamma\,c(d)+c(R)\right)}. \tag{8}$$

With $\eta=1$ and the measured block counts, Eq. (8) gives 2.24× and 2.80× against measured 2.09× and 2.58× (12 problems, $d=\gamma=4$). One factor $\eta=1.07$, fitted to all 24 per-problem timings at once, gives 2.08× and 2.61×, with per-problem errors up to 5.4%. Sequential decoding is equally fast with either cache (ratio 1.00), so the gain of the deepest plane is entirely its higher acceptance.

Figure 5b evaluates Eq. (8) over γ. This is an extrapolation: the flat-acceptance assumption was tested only up to $\gamma=4$, and only $d=\gamma=4$ was timed. In the model, deepest-plane drafts both raise the speed-up and move the best block length up: at $d=4$ the full cache peaks at $\gamma=5$ (1.98×), while the deepest plane still rises at $\gamma=8$ (2.87×).

3.5 Memory and fixed-memory throughput

WaveFront's batched emulator (Ha & Kim, 2026) allocates exactly Eq. (3): over all 26 cells of batches 6 and 8 (sequences of 808 tokens; 0, 1 or 4 shared core slots), the reported cache size matches the formula to its printed precision. Peak reserved memory adds o_0+o_1B, with $o_0=7.2$ GB and $o_1=0.18$ GB (full) or 0.15 GB (one slot) per sequence, fitted on batch 6. This predicts the 9 batch-8 cells, which ran on another node and up to $B=256$, within 0.15% (Figure 3). Under the pre-registered 95 GB budget, the largest batch is $B_{\max}=36$ with the full cache (2.25 GB per sequence) and 302 with one slot.

Throughput per configuration follows $T(B)=B/(a+bB)$, a fixed per-step time plus a per-sequence time (largest residual 2% and 6% for the two series below). The full cache with draft-then-verify saturates at $T_\infty=449$ tokens/s

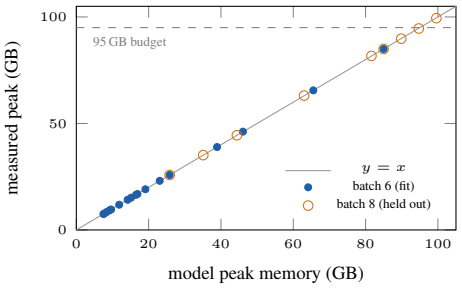


Figure 3. Peak reserved memory of WaveFront's batched emulator against the memory model (Eq. (3) plus $o_0 + o_1 B$ fitted on batch 6). Batch-8 cells (open) ran later on another node and were not used in the fit.

Table 2. KV memory per sequence at $L = 1024$ from Eqs. (3) and (10) (bf16, MB $= 10^6$ bytes), and its fraction of the full cache.

	Huginn-0125		Raven-Llama	
Cache	MB	frac.	MB	frac.
full	2,855	100%	419	100%
deepest plane	173	6.1%	29	7.0%
two planes	260	9.1%	42	10.0%
RIPE, $W = 16$	215	7.5%	35	8.5%
RIPE, $W = 64$	341	11.9%	54	12.8%
RIPE, $W = 256$	843	29.5%	127	30.2%

with half-saturation at $B_{1/2} = 33$; one shared slot with WaveFront's scheduler at 689 tokens/s with $B_{1/2} = 27$ (batch 6). The fixed-memory advantage of sharing has two factors (batch 6, Figure 5c). At the full cache's best batch ($B = 32$), one shared slot already runs 365 tokens/s against the best full-cache configuration's 218, 1.67×; with the same scheduler and acceptance rate the shared slot is 1.63× faster than the full cache at that batch, the cache size being the only difference. Memory then lets the shared slot run $B = 64$, another 1.34×, for 2.24× in all (Section 5.3).

4 RIPE

RIPE (*Recent-window Intact, deepest Plane Elsewhere*) reads the deepest plane for old positions and the own plane for the W most recent ones:

$$\pi_{\text{RIPE}}(r, t, j) = \begin{cases} r, & t - j \le W, \\ R, & t - j > W, \end{cases} \tag{9}$$

$$M_{\text{RIPE}}(L) = M_{\text{deep}}(L) + b\, n_{\text{core}}\, (R-1)\, \min(W, L). \tag{10}$$

Our implementation keeps a deepest-plane array for all positions plus a ring buffer of $W \times (R-1)$ core planes; a position leaving the window drops its non-deepest planes. At $L = 1024$, $W = 64$ costs 341 MB on Huginn (11.9% of 2,855 MB) and 54 MB on Raven (12.8%), i.e. about 168 MB per Huginn sequence over the deepest-plane cache (Table 2).

Algorithm 1 One RIPE self-speculative step (greedy)

Input: cache $\mathcal{C}$ (plane R of all positions; planes $1, \dots, R-1$ of the last W), last token x, draft depth d, block length $\gamma \le W$
for $i = 1$ **to** γ **do**
  $\hat{y}_i \leftarrow \arg\max$ of a d-loop pass on $\hat{y}_{i-1}$ ($\hat{y}_0 = x$); cached positions read plane R
end for
$z \leftarrow$ one R-loop pass on $[x, \hat{y}_1, \dots, \hat{y}_\gamma]$; loop r of token t reads plane r of j if $t - j \le W$, else plane R
$k \leftarrow \max\{i : \hat{y}_{1..i} = \arg\max z_{0..i-1}\}$; emit $\hat{y}_{1..k}$ and $\arg\max z_k$
append the emitted tokens' planes to $\mathcal{C}$; drop planes $1, \dots, R-1$ of positions older than W

Proposition 1 (block-independent verification)**.** *If $W \ge \gamma$, verifying a block of $\gamma + 1$ tokens in one pass under π_{RIPE} computes, in exact arithmetic, the same logits and the same cached keys and values as decoding the tokens one at a time under π_{RIPE}.*

Proof sketch. π_{RIPE} depends only on r and $t - j$. A block token t reads earlier block tokens j with $t - j \le \gamma \le W$, hence plane r, which the block pass has computed by the time it runs loop r. A cached position $j < t_0$ with $t - j \le W$ also satisfies $t_0 - j \le W$, so its plane r is still in the window. All other positions read plane R, which is stored for every position. These are exactly the reads of sequential decoding, so the accepted tokens' cached values agree as well. □

The deepest-plane cache ($W = 0$) violates the premise: a block token cannot read plane R of an earlier block token, which does not exist yet, so batched verification reads plane r there, and outputs depend on where blocks start. Section 5.2 measures how much this matters on Huginn.

Drafts and the choice of W. Drafts do not affect correctness and may read any stored plane; reading plane R everywhere gives them the higher acceptance of Section 3.3. The ripeness test suggests the window length: where the deep loops are ripe to bf16 precision (Raven), $W = \gamma$ suffices for exact verification; where they are not (Huginn), $W = 64$ removed most of the tail in our data (Section 5.1). We did not run a window larger than zero inside the speculative decoder (Section 7).

5 Evaluation

Protocol. Each batch had a pre-registration with decision rules and predictions, a frozen file table (SHA-256), on-node CPU self-tests, a separate unscored Nsight Compute pass, and an archive verified on both ends. All runs used one NVIDIA GH200 (96 GB) with PyTorch 2.14 and CUDA

Table 3. Equal-storage caches, first-decoded-position top-1 agreement with the full cache (batch 2, Huginn, AlpacaEval, bf16). Planes: core planes kept per position.

Cache	Planes	Agreement
deepest plane (`compress-s4`)	1	0.990
two deepest planes (`s8`)	2	0.995
share every 8 loops	4	0.980
first 4 loops (shallow window)	4	0.845
share every 4 loops	8	0.970
first 8 loops (shallow window)	8	0.935

Table 4. RIPE windows on Huginn (batch 8, 48 prompts). Tail: positions with KL>0.1. Extra: memory over the deepest-plane cache at $L = 1024$ (Eq. (10)).

Cache	Tail	Max KL	Flips	Extra MB
deepest plane ($W = 0$)	9	5.05	35	0
RIPE, $W = 16$	5	0.97	37	42
RIPE, $W = 64$	1	0.13	31	168
RIPE, $W = 256$	0	0.04	10	670
$+$ 4 sinks	9	5.03	38	10
two planes (`s8`)	9	5.24	42	87
fp32 full cache	1	0.23	32	—

13.0, bf16 and greedy decoding unless stated, and cuDNN attention disabled from batch 7 on (Section 5.5). Prompts are GSM8K 8-shot problems (Cobbe et al., 2021) and AlpacaEval instructions or reference texts (Dubois et al., 2024). Huginn uses HF revision bb6621b6 and Raven revision a5f6f126, both with the official modeling code unchanged. Appendix A lists the batches.

5.1 Generation fidelity and the tail

Huginn: mostly a precision change, plus a tail. The pre-registered statistic is $\mathrm{KL}(L_0\|\text{deepest})/\mathrm{KL}(L_0\|\text{fp32})$ per bin of generated position with a prompt-paired bootstrap. On 48 confirmatory prompts the label is "exceeds a precision change", through a single bin (ratio 13.8 [2.28, 21.2]) in which one problem carries 75% of the KL; other bins have ratios 1.6–3.3. Over all positions the median KL is lower for the deepest plane than for fp32 (6.4×10^{-5} vs. 8.2×10^{-5}) but the mean is 5.7 times higher, because 1% of positions carry 74% of the KL (fp32: 21%; Figure 4a). The 9 positions with KL above 0.1 (fp32: 1) lie in 4 prompts, 5 of them in one GSM8K problem at nearby positions (Figure 4c). Only 2 of them have a reference top-2 probability margin below 0.1, where 8% of all positions lie: near-ties are over-represented (0.7 expected, 2.7×) but account for a minority of the tail. The largest value, 5.05, is one of those two.

Raven: indistinguishable from a precision change. On the same 72 prompts, Raven's deepest plane is "indistinguishable from a precision change" on both prompt sets (bin ratios 1.04–1.51), its largest KL is 0.020 (fp32: 0.014), no position exceeds 0.1, and it flips the top-1 token at 48 positions against 76 for fp32. We had predicted the opposite, a larger effect than on Huginn; the ripening model accounts for the difference after the fact (Section 3.2).

RIPE windows remove most of Huginn's tail. Batch 8 replayed 48 of these prompts through RIPE caches (Table 4, Figure 4b). The pre-registered count of tail positions falls from 9 to 1 at $W = 64$ (89% fewer; ratio 0.11 [0.00, 0.20], wide because four prompts carry all tail positions), and the largest KL from 5.05 to 0.13, below the fp32 replay's 0.23. On the problem of Figure 4c, $W = 16$ already cuts the peak to 0.97 and $W = 64$ to 0.13. Keeping four attention-sink positions (Xiao et al., 2023) at full depth does nothing, and neither does a second stored plane at similar memory. The 48 reference trajectories were bitwise identical to batch 7's, on another node.

5.2 Self-speculative decoding

Batch 8 ran a real greedy self-speculative decoder on Huginn: depth-4 drafts, four per block, and one 32-loop verification pass per block. With the full cache it accepts 3.34 tokens per verification on GSM8K and is 2.09× faster than sequential decoding; with the deepest plane, 4.47 and 2.58× (Sections 3.3–3.4 model both).

With the deepest plane, batched verification depends on where blocks start (Section 4). The pre-registered test compares verification targets of batched and sequential deepest-plane decoding at identical contexts, against the same comparison for the full cache, where the two are equal in exact arithmetic (an fp64 CPU test confirms it) and differ only by bf16 numerics. The ratio is 1.37 [1.20, 1.62], below the pre-registered threshold of 2 (label "indistinguishable from numerics"; we had predicted 3–30), and 28 of 32 batched trajectories equal the sequential ones. On Huginn the block dependence is therefore small; Proposition 1 removes it by construction.

Using WaveFront's released code (Ha & Kim, 2026) unchanged, batch 5 reproduced its Huginn numbers on GH200: speed-up 2.64× at acceptance 0.878 without sharing and 4.14× at 0.965 with one shared slot.

5.3 Fixed-memory throughput

Batches 6 and 8 placed WaveFront's scheduler and draft-then-verify under a 95 GB budget at the measured acceptance rates (Figure 5c). The best configuration without sharing is draft-then-verify at $B = 32$ (218 tokens/s, batch 6) and $B = 34$ (253 tokens/s, batch 8); with one shared slot, WaveFront at $B = 64$ (487) and $B = 256$ (572), so sharing is 2.24× and 2.26× faster. The two batches ran

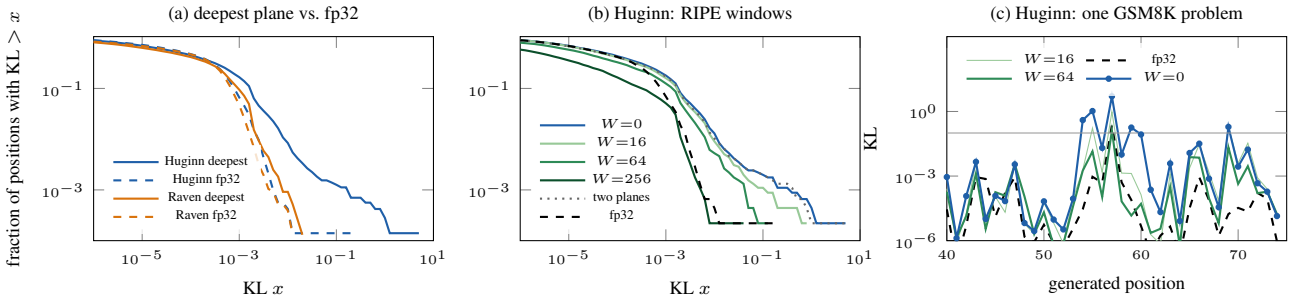


Figure 4. Error of deepest-plane caches in generation (teacher-forced replay of full-cache trajectories). (a) Survival function of per-position KL, 7,130 Huginn and 7,647 Raven positions on the same 72 prompts. (b) RIPE windows on the 48 Huginn prompts of batch 8. (c) KL along the GSM8K problem that holds 5 of Huginn's 9 tail positions.

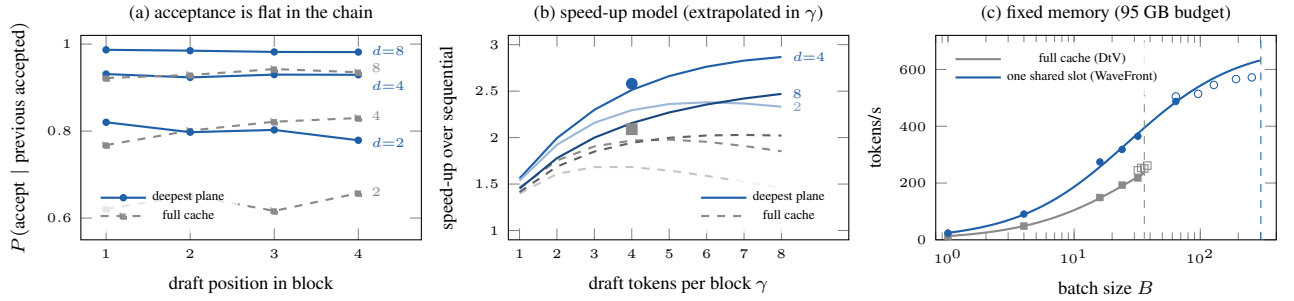


Figure 5. (a) Acceptance at each draft position given that earlier ones were accepted (batch 3, Huginn, GSM8K); flatness supports Eq. (7). (b) Eq. (8) with $\eta = 1.07$ (lighter: $d = 2$, darker: $d = 8$; dashed: full cache); markers: measured speed-ups of the real decoder at $d = \gamma = 4$ (batch 8). Curves beyond $\gamma = 4$ are extrapolations. (c) Throughput of WaveFront's batched emulator at measured acceptance rates; filled: batch 6, open: batch 8 (another node, reported separately); lines: $T(B) = B/(a + bB)$ fitted to batch 6, drawn up to the memory model's $B_{\max}$ at 95 GB (dashed).

on different nodes, where identical anchor cells differed by 2–12%, so they are not pooled. Two pre-registered memory predictions failed (the full cache would run out between $B = 24$ and 32, and at $B = 38$); the memory model of Section 3.5 reproduces both cells' peak memory ($B = 38$ needs 99.4 GB: above the budget, within the device). On Huginn our deepest-plane cache, which needs the same memory as one shared slot, is not lossless relative to the full cache (Section 5.1); we did not check which plane WaveFront's shared slot holds.

5.4 What did not help

Per-head position filtering. The starting plan also proposed keeping, per head, only the prefix positions that a shallow loop attends to. Batch 1 found cross-depth attention affinity real but mostly static (sink plus recent tokens). Batch 8 froze a per-head policy from batch 1 and tested it on 64 unseen AlpacaEval texts at a budget of 10% of the prefix: filtering costs 72 times more KL than deepest-plane storage, the two costs add (ratio 0.88), and the per-head policy is 1.88× worse than top-k for every head [1.65, 2.20]. Selecting at loop 4 instead of 2 cuts the cost to 0.33 of its value, consistent with ripening.

Sinks and a second plane. Neither removed any of Huginn's tail positions (Table 4); a recent window did.

5.5 Reproducibility on GH200

Batch 4 found that the bf16 full-cache reference was not reproducible: a same-process rerun matched bitwise at only 30% of positions, with mean KL 1.6×10^{-4}, about two thirds of the bf16-to-fp32 floor. PyTorch's deterministic mode did not remove it; batch 6b traced it to cuDNN's scaled-dot-product attention (cuDNN 9.24). With that backend disabled, reruns are bitwise identical (100% of positions, across processes and nodes) and decoding is not slower; the deepest-plane labels and ratios were unchanged. Separately, WaveFront's decoder and its autoregressive baseline produce identical texts on 31/40 problems without sharing, with cuDNN disabled or not, and our own full-cache batched and sequential decoders, equal in exact arithmetic, agree on only 20 of 32 trajectories in bf16. Computation shape alone flips bf16 trajectories at this rate, so a "lossless" claim

for speculative decoding in bf16 needs a stated reference computation.

6 Related work

Looped models. Huginn (Geiping et al., 2025), Ouro (Zhu et al., 2025) and retrofitted recurrence (McLeish et al., 2025) train or convert models that iterate a core block. Ouro reports that keeping only the last loop's KV is nearly lossless on its model; we measure when this holds and why it fails on Huginn. WISE (Wan & Chen, 2026) finds that attention routing stabilizes early in depth and reuses the early loops' attention support; we find that prefix keys and values matter late and reuse the last loop's planes. Both exploit convergence across depth, in opposite directions.

Self-speculation in looped models. Huginn's zero-shot self-speculation (Geiping et al., 2025), LoopSpec's pipelined branches (Cho et al., 2026), WaveFront's diagonal schedule with cross-recurrence KV sharing (Ha & Kim, 2026), Mature-Spec's mature values for shallow drafts (Li et al., 2026) and LoopAhead (Liu et al., 2026). Self-speculation by layer skipping in standard models is Draft&Verify (Zhang et al., 2023); the acceptance model is that of Leviathan et al. (2022) and Chen et al. (2023).

KV compression. Cross-loop KV compression and quantization (Neill & Reid, 2026; Kim et al., 2026) and memory-efficient looped inference (Yang & Liu, 2026); for standard transformers, attention sinks and streaming windows (Xiao et al., 2023), quantization (Liu et al., 2024) and paging (Kwon et al., 2023). RIPE's recent window plays the role of a streaming window along depth rather than along the sequence.

7 Limitations and conclusion

Limitations. Two models, one trained recurrent and one retrofitted; the ripening explanation and the ripeness test are post hoc. RIPE with $W > 0$ was evaluated only by teacher-forced replay on Huginn; we did not run it inside the speculative decoder, nor deepest-plane drafts against RIPE verification. The speed-ups come from a batch-1 eager framework and the throughput frontier from WaveFront's emulator; Eq. (8) was timed at one (d, γ). The tail statistics rest on four prompts. Greedy decoding, GSM8K and AlpacaEval only; per-head filtering at one budget and one selection rule.

Conclusion. In looped language models the prefix keys and values that matter are those read by the last loops, and how fast a model's planes ripen decides whether the deepest plane alone is enough. Simple models of sensitivity, ripening, acceptance, cost and memory account for what nine pre-registered batches measured; the memory and acceptance models predict held-out measurements. RIPE keeps the deepest plane everywhere except a short recent window: at 11.9% of the full cache's memory on Huginn it removes most of the deepest plane's error tail, and it keeps batched verification consistent with sequential decoding.

References

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating Large Language Model Decoding with Speculative Sampling. arXiv:2302.01318, 2023.

Cho, S., Cui, L., Kim, S., Han, D., and Han, I. LoopSpec: Pipelined Self-Speculative Decoding for Looped Transformers. arXiv:2609.17184, 2026.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., et al. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168, 2021.

Dubois, Y., Galambosi, B., Liang, P., and Hashimoto, T. B. Length-Controlled AlpacaEval: A Simple Way to Debias Automatic Evaluators. arXiv:2404.04475, 2024.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., et al. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv:2502.05171, 2025.

Ha, H. and Kim, J.-J. WaveFront Decoding: Parallelized Self-Speculative Decoding for Looped Language Models. arXiv:2609.23033, 2026.

Kim, H., Lee, J., Cho, S., Han, D., Han, I., and Kim, S. ResidualQuant: KV Cache Quantization for Looped Transformers with 2-Bit Residuals. arXiv:2610.10381, 2026.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180, 2023.

Leviathan, Y., Kalman, M., and Matias, Y. Fast Inference from Transformers via Speculative Decoding. arXiv:2211.17192, 2022.

Li, G., Su, Z., Yu, H., Jiang, J., Ren, T., Li, Z., et al. Shallow Queries, Mature Values: Depth-Asynchronous Self-Speculation for Looped Transformers. arXiv:2609.34538, 2026.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. arXiv:2402.02750, 2024.

Liu, Z., Xue, N., Bao, R., Xu, Y., Cao, X., Yao, J., and Chen, Z. LoopAhead: Speculating Ahead for Faster Decoding in Looped Language Models. arXiv:2610.21594, 2026.

McLeish, S., Li, A., Kirchenbauer, J., Kalra, D. S., Bartoldson, B. R., Kailkhura, B., et al. Teaching Pretrained Language Models to Think Deeper with Retrofitted Recurrence. arXiv:2511.07384, 2025.

Neill, J. O. and Reid, F. Looped Latent Attention: Cross-Loop KV Compression for Looped Transformers. arXiv:2607.15456, 2026.

Wan, K. and Chen, C. Attention Routing Stabilizes Early: Working-Set Inference for Recurrent Language Models. arXiv:2609.27373, 2026.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453, 2023.

Yang, W. and Liu, S. FlashLoop: Fast and Memory-Efficient Looped Transformers via Lazy Updates. arXiv:2609.29812, 2026.

Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G., and Mehrotra, S. Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. arXiv:2309.08168, 2023.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., et al. Scaling Latent Reasoning via Looped Language Models. arXiv:2510.25741, 2025.

A BATCHES

Each batch had a pre-registration with decision rules and predictions frozen before the run, a frozen file table (SHA-256), on-node CPU self-tests, a separate unscored Nsight Compute pass, and an archive verified at both ends (Table 5).

Table 5. Pre-registered batches, one GH200 each.

Batch	Question	Pre-registered label
1	cross-depth attention affinity	GSM8K trivial; AlpacaEval intermediate
2	which loops' prefix KV matter (Huginn)	deep KV matters
3	deepest plane in generation; acceptance	exceeds precision change
4	reference asymmetry; confirmation; tail	asymmetry negligible; exceeds
5	WaveFront baseline	speed-ups in band
6, 6b	reproducibility; fixed-memory frontier	cuDNN is the source; shared $\geq 1.5\times$
7	batches 3–5 with cuDNN off	reproducible; unchanged
8	batched verification; windows; per-head filtering; frontier	as numerics; window removes most; per-head worse
9	Raven-Llama: region test; deepest plane vs. fp32	deep KV matters; indistinguishable

B PREDICTION LEDGER

Table 6 lists every pre-registered prediction and its outcome, including the ones that failed.

Table 6. Pre-registered predictions and outcomes (all reported, including failures).

Batch	Prediction	Outcome
1	coverage rises with r	held
1	head-median coverage at $r = 4$ in 0.6–0.85	GSM8K above (0.91), AlpacaEval held
1	shallow top-k above the static set by a margin	direction held; magnitude too small ("trivial" on GSM8K)
2	deep KV matters at matched size	held ($e^{6.7}$ bf16, $e^{15.0}$ fp32)
2	`compress-s4` first-position agreement ≥ 0.95	held (0.990)
3	deepest-plane cache indistinguishable from a precision change	failed ("exceeds", via the tail)
3	deepest-plane cache raises acceptance	held (3.31→4.33 at $d = 4$)
4	reference asymmetry changes the ratio by $> 10\%$	failed (0.5–5%)
4	confirmatory set "intermediate"	failed ("exceeds", one prompt)
4	bf16 reruns bitwise identical	failed (30% of positions)
5	speed-up 2–3× unshared, 3.5–5× shared	held (2.64×, 4.14×)
5	WaveFront token-lossless (match ≥ 0.99)	failed (31/40 texts identical)
6	bf16 rerun noise is real	held
6	deterministic mode removes it	failed
6	WaveFront still not lossless in deterministic mode	failed by rule (8/8; chance ≈13%)
6	speed-up falls with batch size	held
6	shared $\geq 1.5\times$ best lossless at 95 GB	held (2.24×)
6	unshared runs out of memory between batch 24 and 32	failed (32 fits)
6b	source is cuDNN attention	held
7	reruns reproducible with cuDNN off	held
7	ratios rise to 2.5–4 with cuDNN off	failed (unchanged)
7	WaveFront 40/40 with cuDNN off	failed (31/40, 31/40)
8	batched deepest-plane verification $\neq$ sequential (ratio 3–30)	failed (1.37, numerics)
8	batched verification closer to the full cache	failed (0.94)
8	tokens per verification 3.0–3.6 (full), $+ \geq 0.5$ (shared)	held (3.34, 4.47)
8	speed-up 1.5–2.3× (full), 2.0–3.0× (shared)	held (2.09, 2.58)
8	window 64 removes most tail positions	held (89%)
8	two planes remove 20–50%	failed (none)
8	per-head ≈ all top-k	failed (1.88× worse)
8	filtering costs $> 3\times$ sharing; costs add	held (72, 0.88)
8	unshared OOM at batch 38	failed (fits)
8	shared/lossless ratio 2.4–2.8	failed (2.26)
9	Raven: deep KV matters, deep coefficient ≥ 3 (bf16)	held (3.8)
9	Raven: deepest-plane first-position agreement ≥ 0.95	held (0.98)
9	Raven: deepest plane "exceeds", more than on Huginn	failed (indistinguishable, no tail)

C FITTED PARAMETERS

All fits are produced by one script from the batch summaries and raw per-position records, and every number in this paper is generated from those files; none is typed by hand (Tables 8, 9; Figure 6).

Table 7. Ripening fits (keys; values in parentheses). ρ and c from fp32, loops 8–31; Δ_∞ fitted to bf16 with ρ, c fixed.

	Huginn	Raven
ρ	0.88 (0.87)	0.81 (0.81)
half-life (loops)	5.2 (5.1)	3.3 (3.3)
Δ_∞ (bf16)	0.0072 (0.0115)	0.0048 (0.0101)
deep-loop Δ, bf16	0.0116 (0.0192)	0.0047 (0.0098)
deep-loop Δ, fp32	0.0097 (0.0162)	1.6×10^{-4} (3.7×10^{-4})
max $\lvert \log$ residual$\rvert$, fp32	0.28	0.51

Table 8. Throughput fits $T(B) = B/(a + bB)$ per configuration of WaveFront's batched emulator (Section 3.5): saturation throughput $T_\infty = 1/b$ (tokens/s), half-saturation batch $B_{1/2} = a/b$ and largest relative residual. Batch-8 series cover a narrow batch range and ran on another node; their parameters are descriptive only.

Batch	Slots	Scheduler	α	B range	Cells	T_∞	$B_{1/2}$	Residual
6	0	AR	—	1–32	5	394	69	4%
6	0	DtV	0.878	1–32	5	449	33	2%
6	0	DtV	0.965	1–32	5	612	34	3%
6	0	WaveFront	0.878	1–32	5	244	13	3%
6	0	WaveFront	0.965	1–32	5	310	11	6%
6	1	AR	—	1–64	6	461	83	4%
6	1	DtV	0.878	1–64	6	494	38	3%
6	1	DtV	0.965	1–64	6	656	39	6%
6	1	WaveFront	0.878	1–64	6	535	30	4%
6	1	WaveFront	0.965	1–64	6	689	27	6%
6	4	AR	—	1–64	6	395	69	6%
6	4	DtV	0.878	1–64	6	473	35	8%
6	4	DtV	0.965	1–64	6	539	29	10%
6	4	WaveFront	0.878	1–64	6	521	29	9%
6	4	WaveFront	0.965	1–64	6	644	24	8%
8	0	AR	—	32–38	4	372	56	2%
8	0	DtV	0.878	32–38	4	380	18	2%
8	0	DtV	0.965	32–38	4	503	18	2%
8	0	WaveFront	0.878	32–38	4	219	8	0%
8	0	WaveFront	0.965	32–38	4	280	7	0%
8	1	AR	—	64–256	5	308	28	2%
8	1	DtV	0.878	64–256	5	443	24	1%
8	1	DtV	0.965	64–256	5	555	20	2%
8	1	WaveFront	0.878	64–256	5	483	18	2%
8	1	WaveFront	0.965	64–256	5	605	14	2%

Table 9. Acceptance model check on the batch-3 simulation ($\gamma = 4$): mean conditional acceptance α_d, its spread across the four draft positions, number of blocks, and tokens per verification predicted by Eq. (7) vs. simulated. Rejection $1 - \alpha_d \approx k\rho_a^d$ fits $k = 0.62$, $\rho_a = 0.76$ (full cache) and $k = 0.42$, $\rho_a = 0.66$ (deepest plane).

Cache	d	α_d	Spread	Blocks	τ model	τ sim.
full cache	2	0.637	0.041	2280	2.46	2.44
full cache	4	0.805	0.062	1676	3.40	3.31
full cache	8	0.932	0.021	1281	4.37	4.34
deepest plane	2	0.800	0.041	1619	3.36	3.41
deepest plane	4	0.928	0.008	1276	4.33	4.33
deepest plane	8	0.984	0.005	1139	4.84	4.85

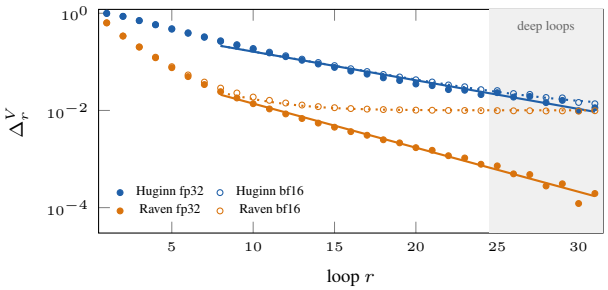


Figure 6. Ripening of prefix values, as Figure 2c shows for keys: fp32 (filled), bf16 (open), geometric fit to fp32 over loops 8–31 (solid) and the same fit with one bf16 floor (dotted).